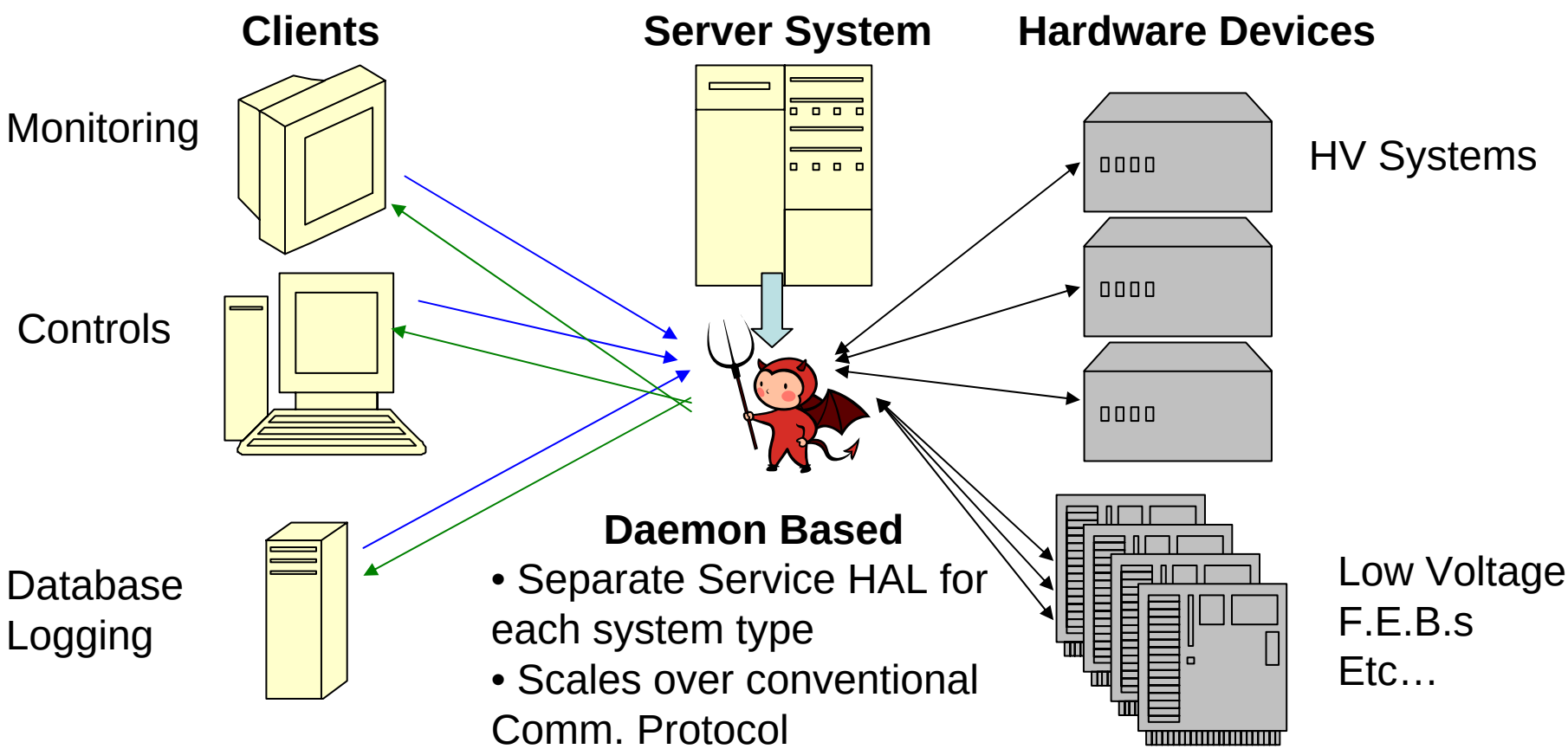
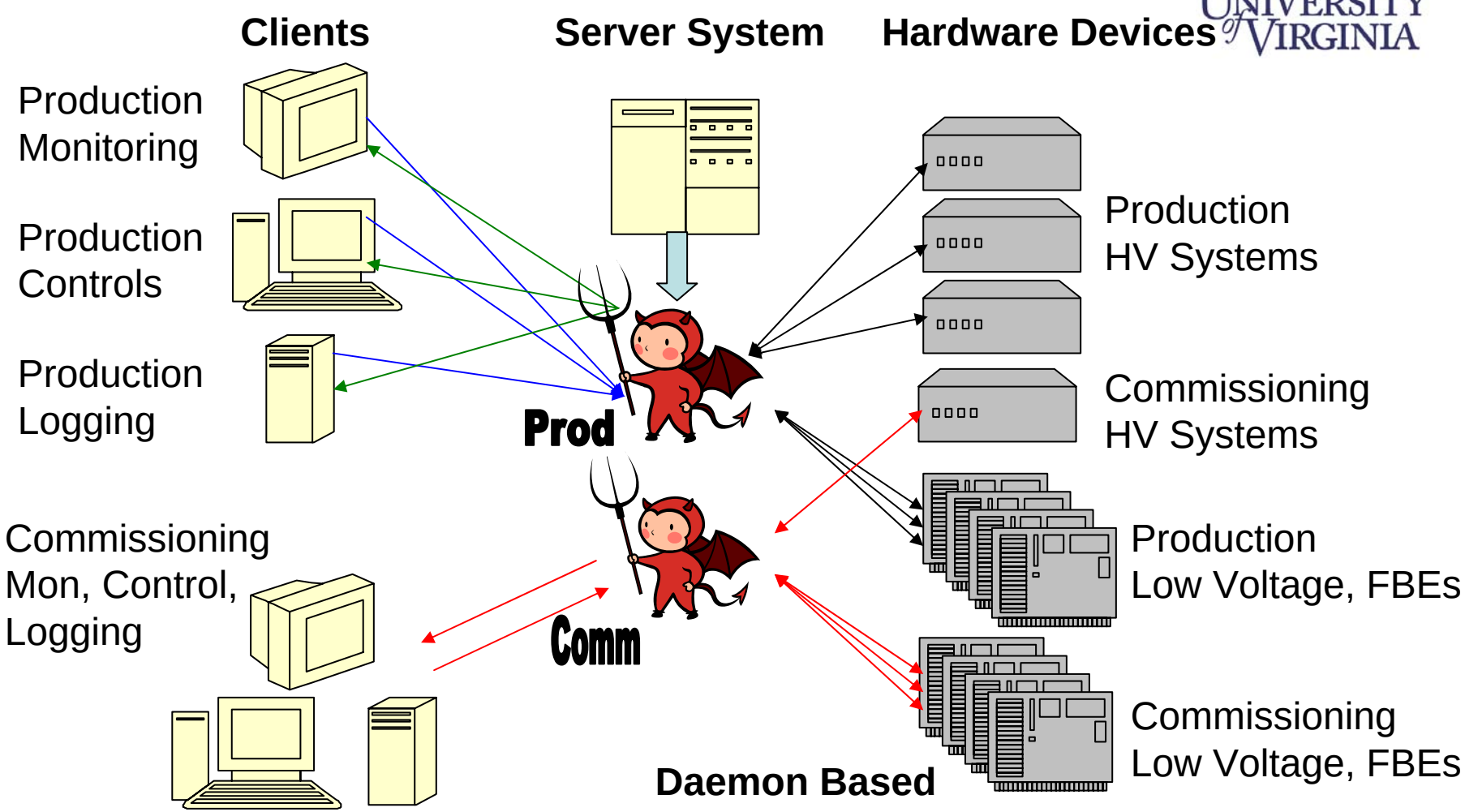
# Detector Control Systems

## EPICS Infrastructure
## And
## Python Interfacing

# Client/Server

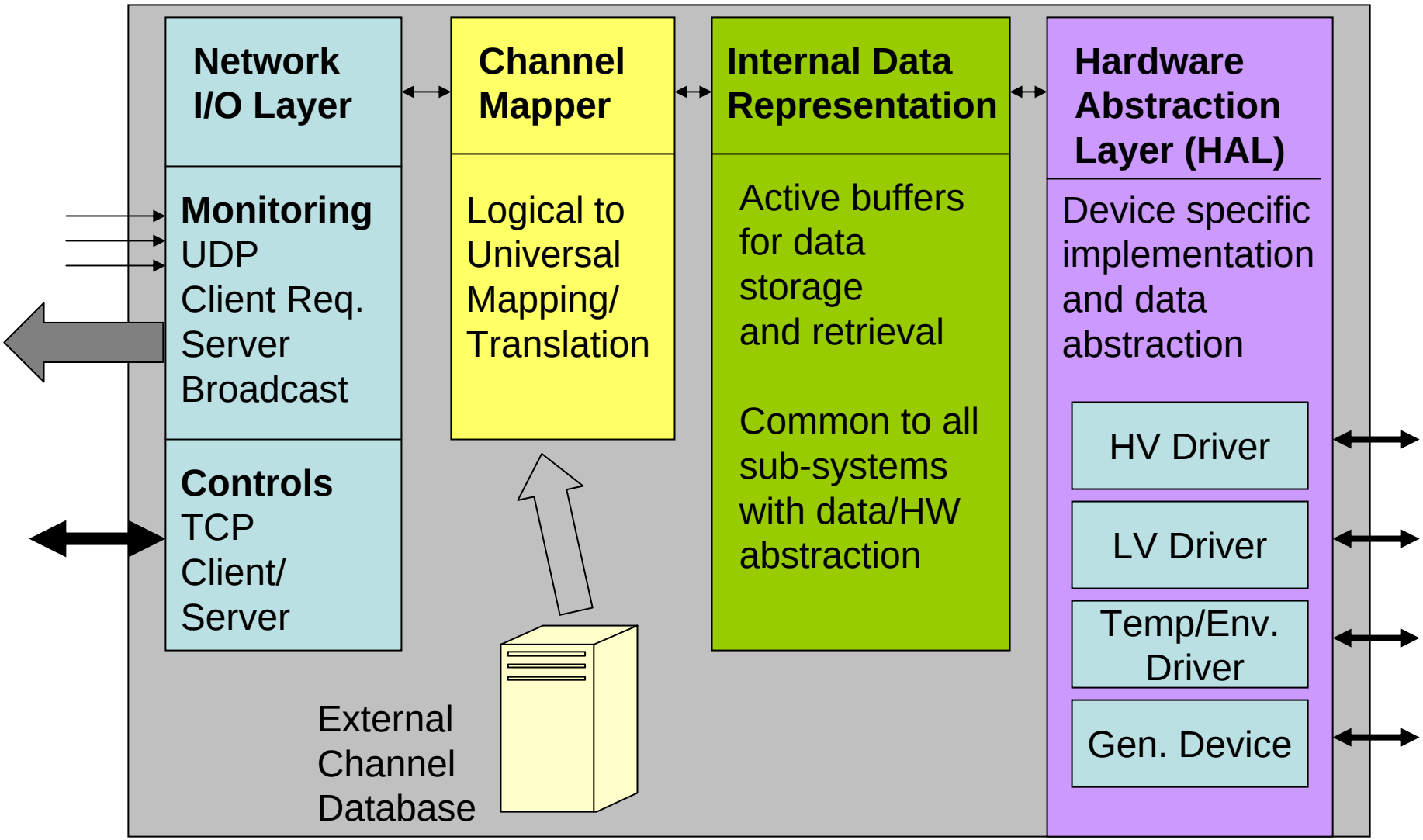


# Production & Commisioning

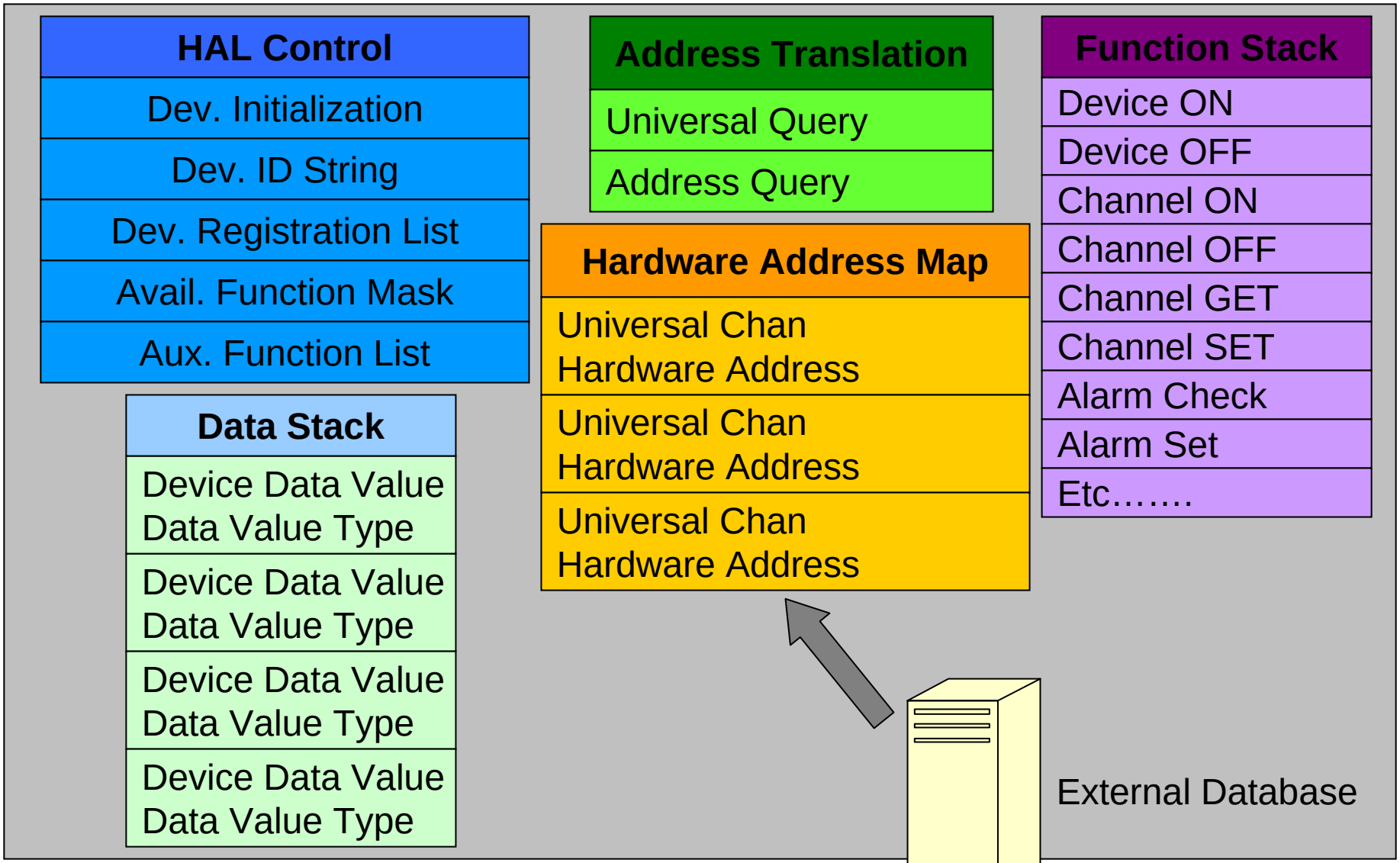



- Separate copies of the daemons run simultaneously with different hardware lists for Production and Commissioning
- Allows for seamless transitions between detector changes
- Common Clients can be used for both tasks

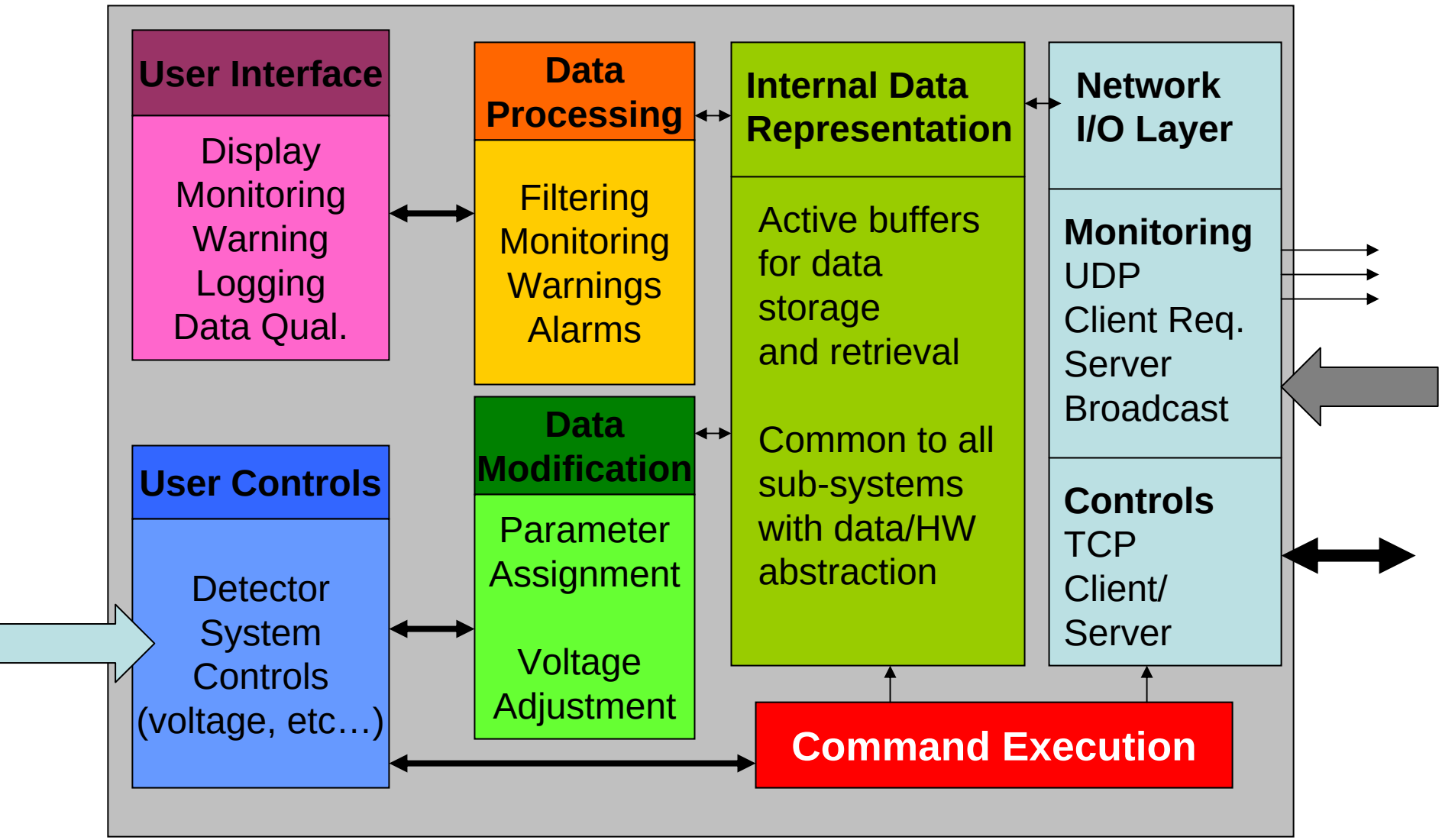
# Daemon Structure



# Example HAL Structure



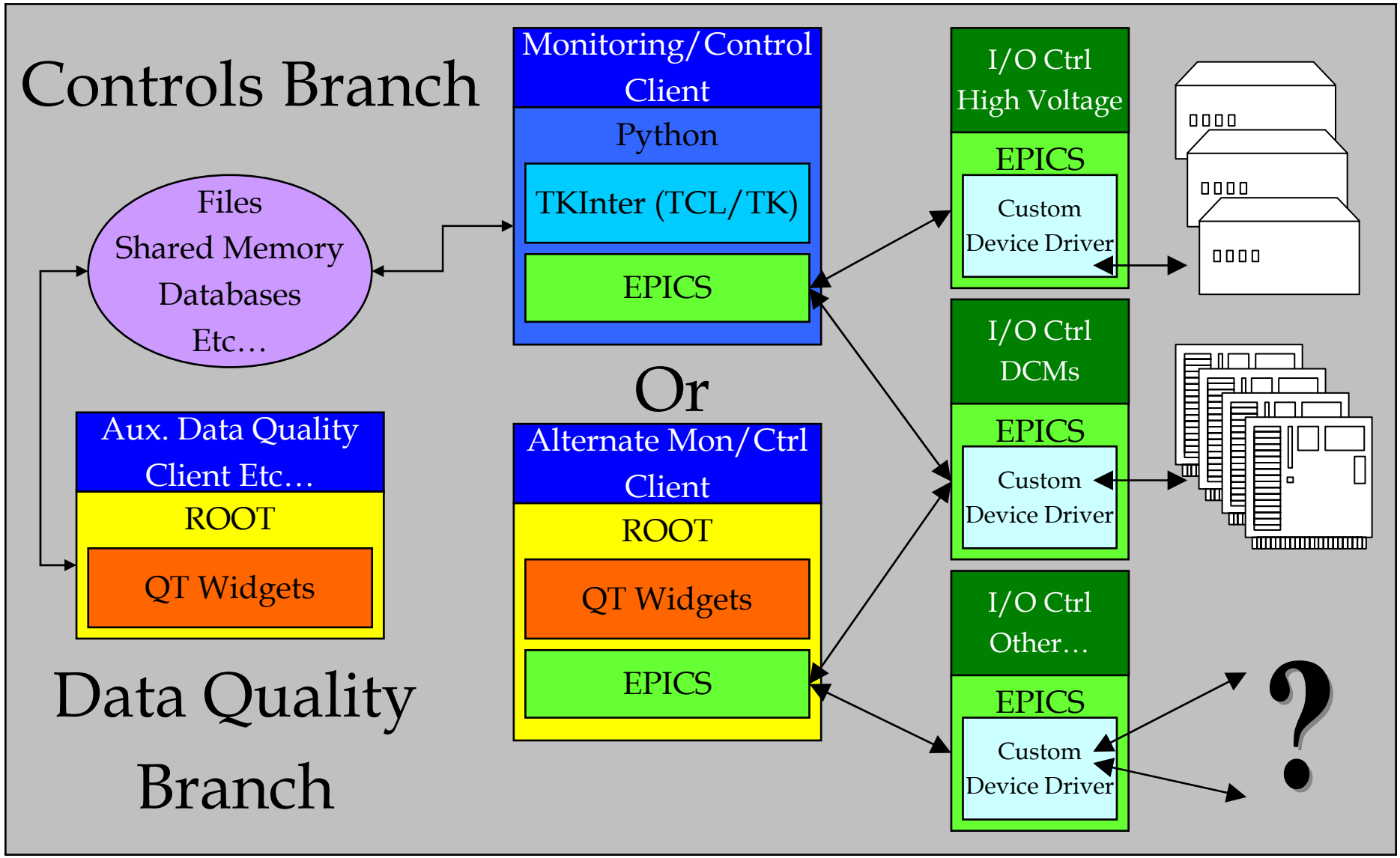
# Client Structure



# Implementation

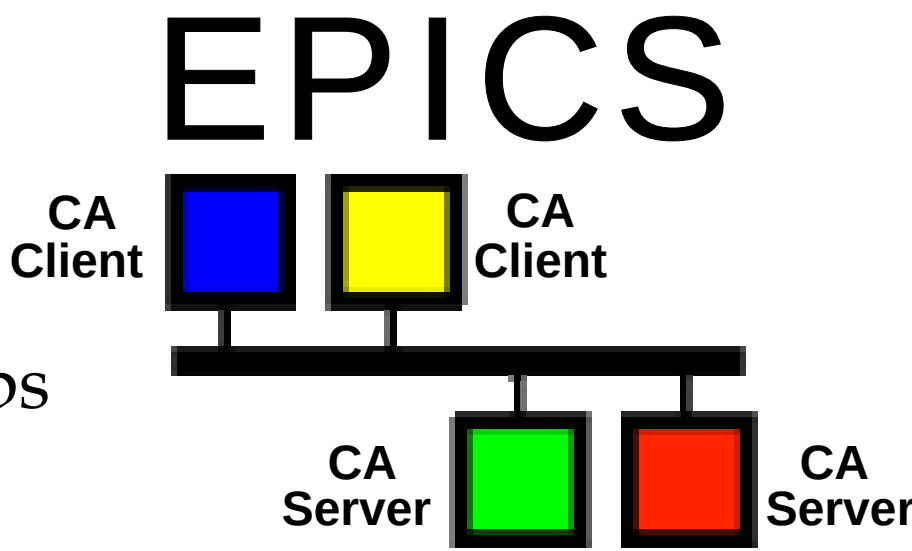
- Choosing an implementation has focused on three external goals in addition to the detector requirements, to:
  - Minimize Cost
  - Minimize Development time
  - Retain flexibility and expandability
- Choosing an existing Detector Controls infrastructure appears to be the best way to meet these.
- Use:
  - EPICS – Detector controls, client/server protocol, internal data representation
  - Python – Cross platform Scripting interface with good EPICS support/hooks for device control
  - TKInter – TCL/TK GUI set for building the graphical interfaces with Python
  - ROOT – Additional GUI and visualization for data quality/monitoring

# Example Implementation



# EPICS Infrastructure

- EPICS (Experimental Physics and Control Systems) developed by Argonne National Lab is based on a server/client model similar to that which we desire for NOvA
- Provides Infrastructure
  - Network Protocol
  - Database handling
  - Data processing
  - Hooks for common apps
    - Python, Perl, C/C++



# EPICS Advantages

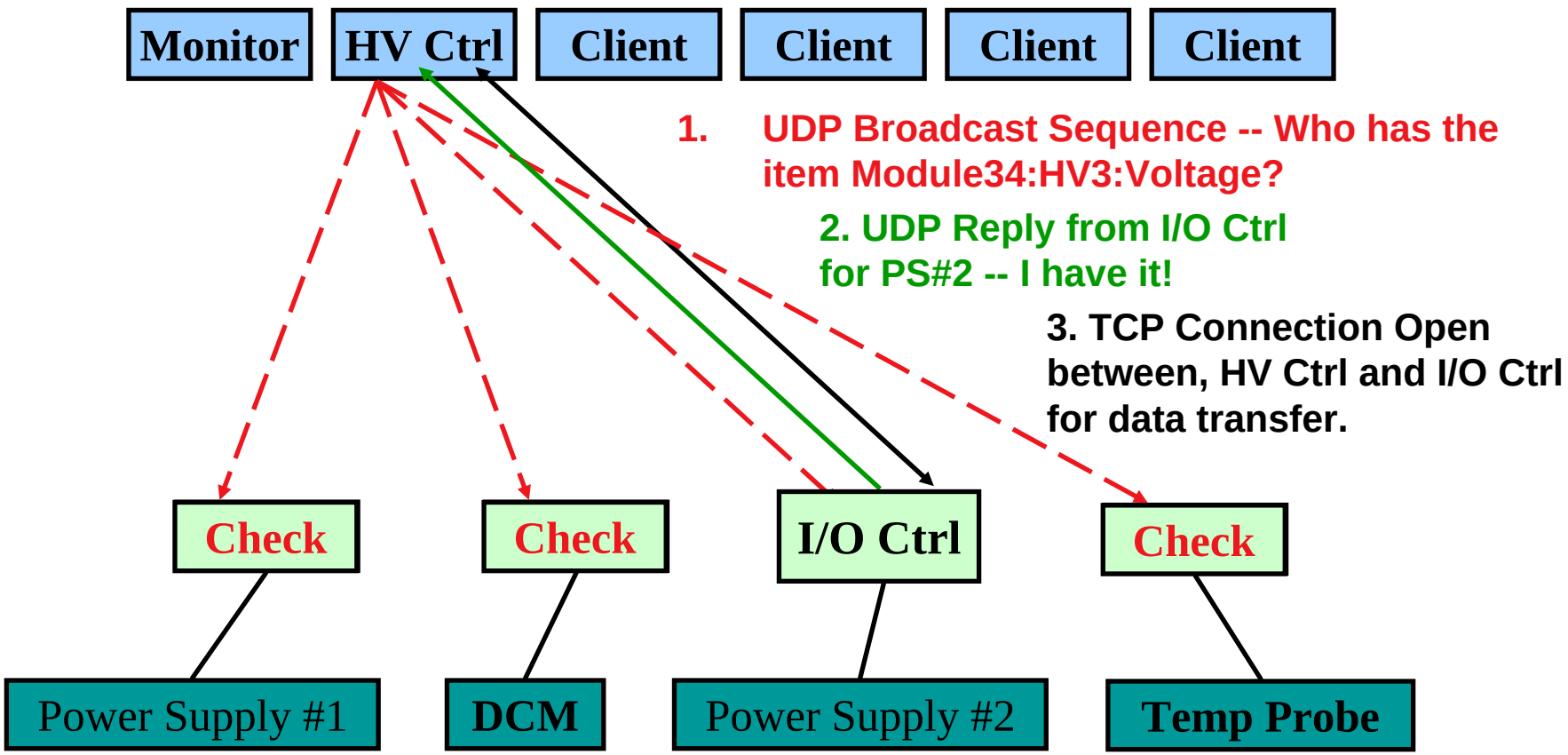

- Cost – It is free!
- Protocols and Database management already developed and well documented.
- Runs on multiple platforms including PC/Linux
- Used in other large scale experiments, and is well supported by the labs.
- Device drivers for certain "common" instruments already exist.
  - i.e. Tek Oscilloscopes, generic CAMAC devices, FNAL beam monitors etc…
- Monitoring and data quality tools already exist and can be adapted
- Tool kits are available to allow for interface of the EPICS base with external packages (i.e. GUI development)

# EPICS Disadvantage


- Requires development of dedicated I/O control drivers for each custom device we want to monitor or control
- Requires all clients and servers to be physically on the same local network (i.e. no direct off-site client access)
- Relies on global broadcasts for client/server communication which complicates partitioning of the experiment into “Production” and “Test” segments
  - But there is docs on how to do this
- I/O Controller setup is targeted more towards direct hardware access than to high latency network access
  - But can write custom IOCs to do this
- Client access is via individual record requests, not large block requests
  - Inefficient for monitoring LARGE numbers of channels
  - Ways around this with custom “record” design etc…

# Channel Access Example

Example:
Get the value of a high voltage channel for monitoring. (Module 34, Channel 3)
Call the channel "Module34:HV3:Voltage" and make the following requests:



# EPICS Performance

- Performance depends upon efficient implementation of device drivers for I/O controllers, and sequencer vs. database operation modes
- Benchmarks from Argonne*:

| Machine | OS | CPU | Speed | Rec/sec | %CPU |
|---|---|---|---|---|---|
| **MVME167** | **vxWorks** | **68040** | **33MHz** | **6000** | **50** |
| **MVME2306** | **vxWorks** | **PPC604** | **300MHz** | **10000** | **10** |
| **MVME5100** | **vxWorks** | **PPC750** | **450MHz** | **40000**** | **10**** |
| **PC** | **Linux** | **PII** | **233MHz** | **10000** | **27** |
| **PC** | **Linux** | **P4** | **2.4GHz** | **50000** | **9** |

This is what we can expect

*Benchmark figures courtesy of Steve Hunt (PSI)
**Extrapolated from performance figures provided by L.Hoff, BNL

# Projected Performance

UNIVERSITY of VIRGINIA

- Assuming device drivers similar to the ANL test setup and hardware access times/topology, we can expect:
  - ≈ 50,000 data values processed per server per second.
  - Assumes we want to retain a "safe" cpu load (10-20% average)
  - Assumes EPICS operating in simple database mode

- More realistic – Implementation of EPICS control systems at DØ
  - Central Fiber Tracker (CFT):
    - 1 channel server per 20 DFEAII boards
    - Run on 1GHz processor linux computer using gigabit fiber to access crate
    - Monitor and control ≈ 800 values @ 1Hz with 2% cpu load*
  - Simple linear scaling up to 50k variables and a 3GHz processor
    - Expected cpu load ≈ 42%

Note: This is "database" mode not monitoring state machine mode

**CPU load average per 50k data ≈ 10-50%**

*Average cpu ussage, actual load spikes with access operations

# Monitoring Load

| System | Values | Channels | Total |
|---|---|---|---|
| Low Voltage | 6 | 81 | 486 |
| High Voltage | 2 | 162 | 324 |
| DCMs | ≈ 20 | 324 | 6480 |
| Water Cooling | ≈ 16 | 144 | 2304 |
| Environmental | 100-400 | | 100-400 |
| FEBs (via DCMs) | 10-20 | 20,000 | 400k |
| | | TOTAL | 9984<br>410k w/FEBs |

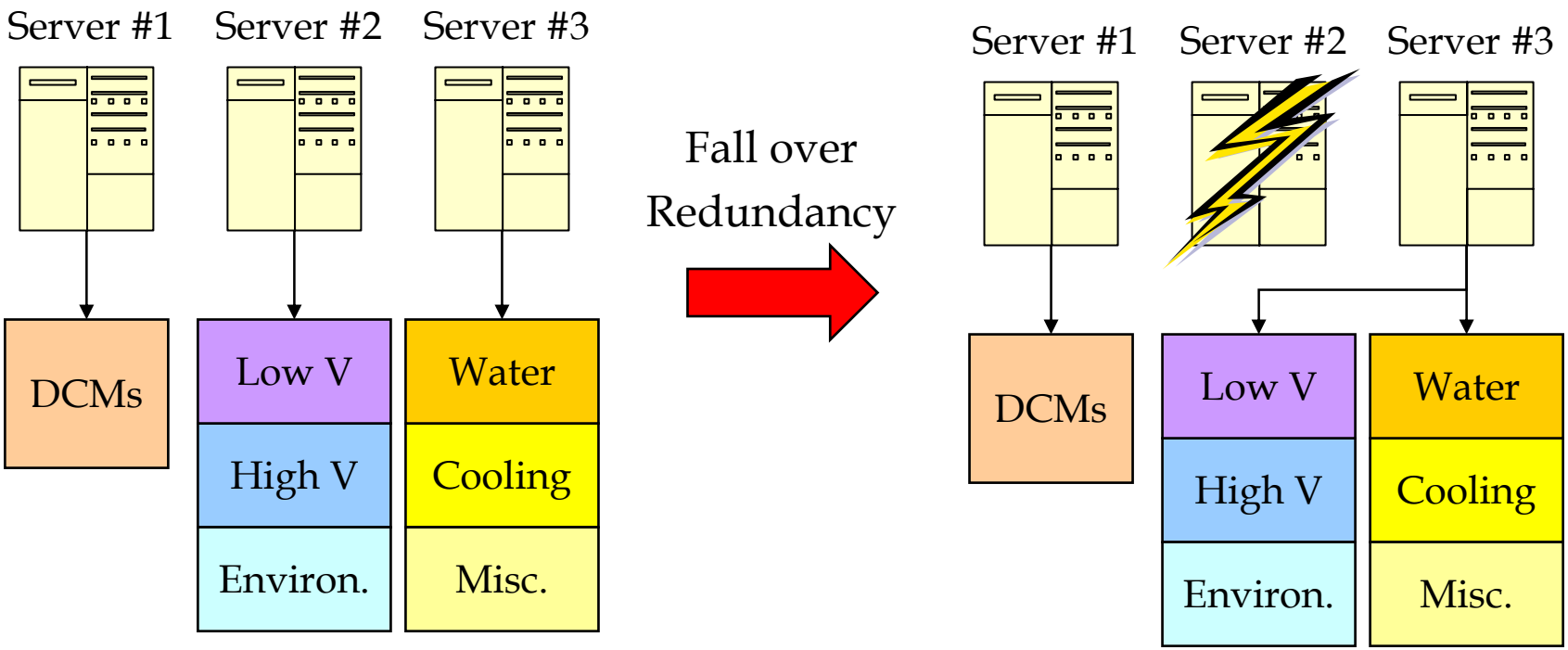
- Monitoring load is computed both with and without individual FEB operational parameters included in the monitoring stream.
- Bandwidth per monitoring cycle from raw devices to channel access servers (w/o FEBs) should be ~1MByte after overhead

# Monitoring Cycle


- Without FEB monitoring, we expect to control and monitor on the order of 10k operational parameters.
- We can use the EPICS state machine functionality instead of the simple database records and remain within the CPU budget
- If the hardware allows, it will be possible to readout and perform **continuous** state monitoring at the channel server level (e.g. 1Hz monitoring cycle)
- Periodic state reporting between client and server can be scheduled for database recording, trend plotting, data quality analysis etc…
- Detection of faults can be reported immediately to the monitoring clients instead of waiting for a client initiated request on a slow cycle

# Computing Resources

- It should be possible to monitor the base 10k operational parameters from one server
- For fault tolerance we should break the load between multiple servers, each servicing a subset of the monitoring subsystems, and configured to provide fall over redundancy



# FEB Monitoring in EPICS

- If we include the FEB operational parameters then we need to expand the number of monitoring nodes to accommodate the load
- At a monitoring frequency of 1Hz this means 400k parameters
- This means 8 monitoring stations minimum, 10 with double fault fall over redundancy

Options:

- Can reduce the monitoring frequency
- Level the FEB monitoring/status information in the primary data stream

| Montoring | Channels | Base Computing | Fall Over Redundancy | Total |
|---|---|---|---|---|
| Base Op. Parameters | 10k | 1 | Double fault | 3 |
| Base + FEBs | 410k | 9 | Double fault | 11 |

# Computing Costs

- Monitoring server requirements are based on a system capable of processing 50k records per second
  - 3 GHz processor class Linux PC
  - Large (2 GB) system memory to aid internal database speed
  - Gigabit network
  - 1U rack mount

| Monitoring | Monitoring Servers | Cost Per Station | Total Cost |
|---|---|---|---|
| Integration Prototype | 1 | 2200 | 2.2k |
| Base Op. Parameters | 3 | 2200 | 6.6k |
| Base + FEBs | 11 | 2200 | 22k |

# Monitoring Clients


- Clients are easier!
- Each client can be a dedicated interface to a set of EPICS controlled parameters
- Clients are written in Python with EPICS libraries and TK widget sets for graphical elements (this is similar to DØ)
- This makes clients:
  - Portable and platform independent
  - Easy to modify and maintain
  - Gives reasonable performance
- Clients required to do more intensive processing are written in in C++ using the EPICS libraries and ROOT interface/widgets for visualization
- Logging and database operation can be done in either model using the standard C++ or Python interfaces to MySQL etc…

# Client Development


- Monitoring clients can be developed independent of the channel servers because they use the EPICS protocol and standard calls for communications
- This means client and server development can begin in parallel
- Client/Server integration testing can be performed with mock servers that used “dummy” device drivers to generate data streams
  - This means software development can begin prior to hardware acquisition
  - Software is insulated from hardware changes

# Server Development

- The channel servers (I/O controllers) need to be written in C/C++ with EPICS libraries.
- Custom device drivers will have to be developed for each system we wish to monitor. This means:
  - High Voltage system (CAEN)
  - Low Voltage systems (Wiener)
  - Data Concentrators Modules
  - Water and Cooling
  - Environmental
- Most of these systems will be capable of communicating over Ethernet using IP, which will simplify driver development, but each system will need a dedicated driver.
- The server infrastructure is independent of the device and can be developed without the hardware
- The device drivers NEED example hardware for development and testing
- For the integration prototype this means we need to know the hardware decisions with some lead time to have fully functional monitoring in place

# Client/Server Resources (Development)

- Monitoring/Control client and server development can proceed in parallel

| Task | Time Span | Personnel | FTE |
|---|---|---|---|
| Integration Prototype Monitoring clients and servers | 1.0 | 1 | 1.0 |
| Production monitoring, Control, Logging clients | 1.5 years | 0.5 | 0.75 |
| Production Channel servers (HV, LV, DCM, FEBs, etc…) | 2.5 years* | 0.25 | 0.75 |
| Continued Maint, Service, Updates | 5 years | 0.125 | 0.625 |

*Hardware development dependent